JMLR: Workshop and Conference Proceedings 29:483–497, 2013 ACML 2013

EPMC: Every Visit Preference Monte Carlo for Reinforcement Learning

Christian Wirth CWIRTH@KE.TU-DARMSTADT.DE
TU Darmstadt, Germany
Johannes Fürnkranz FUERNKRANZ@KE.TU-DARMSTADT.DE
TU Darmstadt, Germany

Abstract

Reinforcement learning algorithms are usually hard to use for non expert users. It is required to consider several aspects like the definition of state-, action- and reward-space as well as the algorithms hyperparameters. Preference based approaches try to address these problems by omitting the requirement for exact rewards, replacing them with preferences over solutions. Some algorithms have been proposed within this framework, but they are usually requiring parameterized policies which is again a hinderance for their application. Monte Carlo based approaches do not have this restriction and are also model free. Hence, we present a new preference-based reinforcement learning algorithm, utilizing Monte Carlo estimates. The main idea is to estimate the relative Q-value of two actions for the same state within a every-visit framework. This means, preferences are used to estimate the Q-value of state-action pairs within a trajectory, based on the feedback concerning the complete trajectory. The algorithm is evaluated on three common benchmark problems, namely mountain car, inverted pendulum and acrobot, showing its advantage over a closely related algorithm which is also using estimates for intermediate states, but based on a probability theorem. In comparison to SARSA(λ), EPMC converges somewhat slower, but computes policies that are almost as good or better.

Keywords: Preference Learning, Reinforcement Learning, Monte Carlo, Every Visit

1. Introduction

Preference-based reinforcement learning (PBRL) is concerned with algorithms that do not assume the availability of a reward signal, but only preferences about policies, trajectories, states or actions. This means a numeric scale for the feedback is not required. Especially for problems with unreliable rewards, like medical treatment experiments with arbitrary penalties for the death of a patient (Zhao et al., 2009), this is a significant advantage. Preference-based approaches also do not assume intermediate feedback after each state transition. This means PBRL algorithms in general can be deemed simpler to use than classic reinforcement learning approaches, due to the simplified feedback. Additionally, preference learning comes with the promise of solving problems which can not be represented with a single, numeric reward in the future. Admitting incomparability as preference predicate would enable learning from structures, not representable by a single objective.

Currently, there are two main driving directions within the PBRL community (Wirth & Fürnkranz, 2013). On the one hand, several approaches search within the policy space using a preference relation over policies that is either directly given or inferred from preferences between trajectories, states, or actions (Akrour et al., 2011; Wilson et al., 2012). This results in a ranking of policies which can be used for creating an improved policy by utilizing evolutionary strategies or Bayesian approaches. The major drawback is that all of the above-mentioned approaches require parameterizable policies. This can be a severe problem for applying these algorithms to real-world problems, especially for non-expert users. On the other hand, value-based approaches assume that the preference feedback is based on some kind of stationary utility or reward function, enabling the computation of a utility value for each state-action pair. This setting is quite similar to inverse reinforcement learning (Abbeel and Ng, 2004; Abbeel & Ng, 2010), but with the main difference that the agent may also observe preferences between suboptimal policies. This scenario can be tackled with Monte Carlo techniques. The advantages of this approch are that it is model-free and that there is no need for parameterizable policies.

We build upon the work of Fürnkranz et al. (2012) who introduced a first Monte Carlo-based approach to this problem, but this algorithm is not capable of updating multiple states with a single preference statement. Each preference feedback is only used for a single state-action-action comparison. The *Every Visit Preference Monte Carlo (EPMC)* algorithm presented here utilizes a relative Q-function, defining the expected gain of picking an action a compared to another action a'. This function can be estimated for subtrajectories, enabling the reuse of preferences for updating intermediate states. Most closely related to our approach is the work of Wirth and Fürnkranz (2013), who proposed an algorithm utilizing multiple state updates, which uses a probability theorem instead of an relative Q-function.

Section 2 presents the definition of the underlying Markov process, information about the used preferences and some required assumptions for the new algorithm. The problems of preference-only feedback for every-visit Monte Carlo methods and how they are tackled, is the topic of section 3. The resulting EPMC algorithm is presented in sec. 4. This algorithm has been evaluated in the mountain car, inverted pendulum and acrobot domains, as defined in sec. 5. Section 6 shows that EPMC is greatly outperforming the algorithm of Wirth and Fürnkranz (2013). In a comparison with SARSA(λ), it achieved policies nearly as good or better but with slower convergence rates for two out of the three domains, while using the same amount of sampled trajectories. Before concluding the paper (sec. 9), some information about related and future work is presented (sec. 7 & 8).

2. Preliminaries

2.1. Markov Decision Process without Rewards (MDP\R)

An (finite-state) MDP\R, as introduced by Abbeel and Ng (2004), is defined by a quadruple (S, A, δ, γ). Given are a set of *states* $S = \{s_i\}$, *actions* $A = \{a_j\}$, and a probabilistic *state transition function* $\delta : S \times A \times S \to [0, 1]$ such that $\sum_{s'} \delta(s, a, s') = 1$ for all $(s, a) \in S \times A$. $A(s)$ denotes the set of actions that are possible in state s, i.e., $A(s) = \{a \in A | \sum_{s' \in S} \delta(s, a, s') > 0\}$. $\gamma \in [0, 1)$ is a discount factor. A reward function $r(s, a)$ is assumed to exist, but not known.
We can therefore make use of the (optimal) state-action function $Q^*(s, a)$ and state value

function $V^*(s)$, as defined by Sutton and Barto (1998):

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} \delta(s,a,s')(r(s,a) + \gamma \max_{a \in A(s)} Q^*(s',a')) \\ V^*(s) &= \max_{a \in A(s)} Q^*(s,a) \end{aligned} \tag{1}$$

$S_0 \subseteq S$ defines the set of valid start states. Only states within this collection can be used to initialize the MDP\R. Additionally, there exists a set $S^F \subseteq S$ s.t. $\forall s \in S^F; A(s) = \emptyset$. This is the set of absorbing or terminal states where no action is possible. In most domains, these are states where the task has been accomplished or where it is impossible to reach an acceptable solution. $\pi : S \times A \to [0,1]$ denotes a policy by defining the probability of selecting action a in state s where $\sum_{a' \in A(s)} \pi(s,a') = 1$.

2.2. Preferences

We assume that the agent receives feedback in the form of trajectory preferences $T_i \succ T_j$. A trajectory is an collection of state-action pairs, commonly represented by $T = \langle s_0, a_0, s_1, a_2, ..., s_{n-1}, a^{n-1}, s_n \rangle, s_0 \in S_0, s_n \in S^F$. This is an ordered list with the possibility $(s_i, a_i) = (s_j, a_j), i \neq j$ and $|T|$ as the amount of state-action pairs in this list. The preference $T_i \succ T_j$ indicates that trajectory T_i is deemed more useful. This means, the (decayed) sum of rewards encountered by the preferred trajectory can be assumed to be higher than for the dominated one. The value of a trajectory T can now be defined as

$$V(T) = \sum_{t=0}^{|T|-1} \gamma^t r(s_t, a_t) \tag{2}$$

which allows to intepret a trajectory preference as

$$T_i \succ T_j \Leftrightarrow V(T_i) > V(T_j) \Leftrightarrow V(T_i) = V(T_j) \cdot x_{i,j} \tag{3}$$

where $x_{i,j}$ is the relative improvement of T_i over T_j. The predicate $T_i \succ T_j$ is equivalent to $x_{i,j} > 1$ for $r : S \times A \to \mathbb{R}^+$. Feedback in the form of $T_i \prec T_j$ is used as $T_j \succ T_i \Rightarrow x_{j,i} > 1$, meaning a preferences defines the relations where $x > 1$ is valid. When required, differences to a reward space $r : S \times A \to \mathbb{R}^-$ are described within the according sections. Here, the difference is $T_i \succ T_j \Rightarrow x_{i,j} < 1$. It should be noted, that comparable trajectories always start in the same state $s_0 \in S_0$ because in several domains, it is not possible to determine a preference for trajectories not fulfilling this requirement. E.g. consider minimizing the driving distance to a certain city. If starting from to different positions, it is possible that one route is making a larger detour, but it seems to be preferred because it is shorter in total.

2.3. Assumptions

The algorithm presented here is subject to two assumptions. First, we assume a reward distribution with low variance, i.e., (2) can be approximated as

$$V(T) \approx \sum_{t=0}^{|T|} \gamma^t \cdot \bar{r}_T \tag{4}$$

where $\bar{r}_T$ is the (unkown) mean of all values $r(s, a), (s, a) \in T$, with T as a trajectory of any length. Of course, all tasks that can be reduced to finding the shortest or longest trajectory through the state space fulfill this requirement. They can be described by a fixed step penalty or reward. Additionally, terminal only reward problems are also having low variance. In fact, a large amount of real world tasks does fit into this framework:

- Game Playing - Each move has a reward of 0 with a terminal reward depending on the outcome of the game.
- Planning - Every step of the plan is penalized with all states fulfilling the task as terminal states. This results in the shortest, successful plan having the highest reward.
- Navigation - A uniform time discretization for the state space enables the usage of a fixed step penalty.

Additionally, the (unknown) rewards are assumed to be *either* all positive ($r : S \times A \to \mathbb{R}^+$) *or* all negative ($r : S \times A \to \mathbb{R}^-$). This results in $x_{i,j} \in \mathbb{R}^+$. This way, it is possible to assume that a higher $x_{i,j}$ value is always preferable for the positive case (or a lower one in the negative case), because it results in a higher reward. As long as the reward space is finite, it is always possible to fulfill this requirement by adding or subtracting a constant factor to the rewards.

3. The Monte Carlo Every Visit Method

Monte Carlo methods can be used to determine $Q^*(s, a)$ by applying policy iteration algorithms. This is achieved by iteratively calculating an approximation $Q_i(s, a)$ of the average return following (s, a) based on all samples collected after i iterations. The return is the sum of decayed rewards $\sum_{t=0}^{|T|-1} \gamma^t r(s_t, a_t), (s_t, a_t) \in T$. The resulting Q function is then used to determine the policy π_{i+1}, which is optimal according to $Q_i(s, a)$. Typically, policy iteration converges towards $Q^*(s, a)$, if it is always possible to sample all state-action pairs. Another aspect concerns the sampling method used. In all cases, a random state out of S_0 is picked, following $\pi(s, a)$ afterwards till a state $s \in S^F$ is reached. The *every-visit* method considered here (see algorithm 1) uses the average of returns following *every* occurrence of (s, a) in a trajectory, as opposed to *first-visit* methods that only average over the *first* occurrences of each state-action pair. This means after sampling a trajectory T for the current iteration, the return following each encountered (s, a) is collected and averaged to update $Q(s, a)$. These Q-values are then used to determine the next sampling policy. Note that the $\arg\max_a Q(s, a)$ policy is a pure exploitation policy, but exploration can be included by utilizing an ϵ-greedy strategy or comparable approaches.

As mentioned in section 2.2, $r(s, a)$ is unknown and we only get feedback in the form of $x_{i,j} > 1$ s.t. $V(T_i) = V(T_j) \cdot x_{i,j}$, which results in several problems. At first, our feedback is limited to information about $x_{i,j}$, which can not be used to calculate $V(T_i)$ or $V(T_j)$ without knowledge about the value of either of these. Secondly, the exact value of $x_{i,j}$ is also unknown. Additionally, we only have a (relative) reward sum sample concerning initial state-action pairs (s_0, a_0) due to $V(T) = \sum_{t=0}^{n-1} \gamma^t r(s_t, a_t), (s_t, a_t) \in T$, meaning we do not have any information about other timesteps. But we also need information about $(s_t, a_t), t \neq 0$ for applying every-visit methods. This means, an approximation for $V(T_i^{\dashv}) =$

```
input  : initial policy $\pi_0$, start states $S_0$, iteration limit $m$
output: improved policy $\pi_m$

Returns$(s,a) = \emptyset$
$Q(s,a) \leftarrow$ arbitrary
for $i = 0$ to $m$ do
    $s_0 = $ RANDOM$(S_0)$
    T = SAMPLEPOLICY$(s_0, \pi_i)$ \\applies $\pi_i$ till $s_n \in S^F$ is reached
    for $t = 0$ to $|T|$ do
        Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) \cup \sum_{k=t}^{|T|-1} \gamma^{k-t} r(s_k, a_k)$
        $Q(s_t, a_t) \leftarrow$ CREATEAVERAGES(Returns$(s_t, a_t)$) \\average return sum
    end
    forall the $s \in S$ do
        $\pi_{i+1}(s) = \arg\max_a Q(s,a)$
    end
end
```

Algorithm 1: Every visit Monte Carlo Policy Iteration

$V(T_j^{\dashv}) \cdot y_{i,j}$ s.t. $T_i = T_i^{\dashv}.T_i^{\vdash}, |T_i^{\dashv}| \neq 0, T_j = T_j^{\dashv}.T_j^{\vdash}, |T_j^{\dashv}| \neq 0$ is required. $T = T^{\dashv}.T^{\vdash}$ denotes that T is a concatenation of two partial trajectories $T^{\dashv}$ and $T^{\vdash}$. $x_{i,j}$ and $y_{i,j}$ are comparable, but $x_{i,j}$ always concerns the complete trajectories T_i and T_j.

In short, our problems are

1. $V(T_i)$ and $V(T_j)$ are unknown.
2. the exact value of $x_{i,j}$ is unknown.
3. $y_{i,j}$ s.t. $V(T_i^{\dashv}) = V(T_j^{\dashv}) \cdot y_{i,j}$ is unknown.

This section explains the problems in detail as well as the applied solutions.

3.1. A relative Q-function

Without a specific reward sum as feedback, it is not possible to estimate $Q(s,a)$ directly. According to section 2.2, we can assume feedback $x_{i,j} > 1$ s.t. $V(T_i) = V(T_j) \cdot x_{i,j}$. $V(T_i)$ is the decayed sum of rewards following the first state-action pair (s_0, a) of trajectory T_i which is a sample for $Q(s_0, a)$ according to the Monte Carlo policy iteration framework. (With $V(T_j)$ as sample for (s_0, a')) Hence we can also define $Q(s_0, a) = Q(s_0, a') \cdot x_{i,j}, (s_0, a) \in T_i, (s_0, a') \in T_j, s_0 \in S_0$. Therefore, it is possible to only store a value $Q_{\text{rel}}(s_0, a, a')$ as relative Q value for a comparison of $Q(s_0, a)$ and $Q(s_0, a')$ We are using $Q_{\text{rel}}(s_0, a, a')$ s.t. $Q(s_0, a) = Q(s_0, a') + Q_{\text{rel}}(s, a, a') \cdot Q(s_0, a')$ with $Q_{\text{rel}}(s_0, a, a') = x_{i,j} - 1$, as shown in the following equation.

$$\begin{aligned}
& Q(s_0, a) = Q(s_0, a') \cdot x_{i,j} \\
\Leftrightarrow\ & Q(s_0, a) = Q(s_0, a') + Q(s_0, a') \cdot (x_{i,j} - 1) \\
\Leftrightarrow\ & Q(s_0, a') + Q_{\text{rel}}(s, a, a') \cdot Q(s_0, a') = Q(s_0, a') \\
& \qquad\qquad + Q(s_0, a') \cdot (x_{i,j} - 1) \\
\Leftrightarrow\ & Q(s_0, a, a')_{\text{rel}} = x_{i,j} - 1
\end{aligned} \tag{5}$$

This form was chosen to gain a Q function that is symmetric due to only storing the additive (or subtractive) part of the relative value.

This training information is also used to determine the inverse Q in the form of $Q_{\text{rel}}(s_0, a', a)$ s.t. $Q(s_0, a') = Q(s_0, a) + Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a)$. By inserting $Q(s_0, a) = Q(s_0, a') + Q_{\text{rel}}(s_0, a, a') \cdot Q(s_0, a')$ into $Q(s_0, a') = Q(s_0, a) + Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a)$ and utilizing $Q(s_0, a) = Q(s_0, a') \cdot x_{i,j}$, we have determined the computation required for reusing the inverse, as described by (6).

$$\begin{aligned}
&Q(s_0, a') = Q(s_0, a) + Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a) \\
\Leftrightarrow &- Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a) + Q(s_0, a') = Q(s_0, a) \\
\Leftrightarrow &- Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a) + Q(s_0, a') = Q(s_0, a') \\
&\qquad\qquad + Q_{\text{rel}}(s_0, a, a') \cdot Q(s_0, a') \\
\Leftrightarrow &\, Q_{\text{rel}}(s_0, a', a) = Q_{\text{rel}}(s_0, a, a') \cdot \left(-\frac{Q(s_0, a')}{Q(s_0, a)}\right) \\
\Leftrightarrow &\, Q_{\text{rel}}(s_0, a', a) = Q_{\text{rel}}(s_0, a, a') \cdot \left(-\frac{1}{x_{i,j}}\right)
\end{aligned} \tag{6}$$

Both, $Q_{\text{rel}}(s_0, a, a')$ and $Q_{\text{rel}}(s_0, a', a)$, are stored separately. An alternative would be to store $x_{i,j}$ instead of $Q_{\text{rel}}(s_0, a', a)$, but this way we can directly aggregate the samples, because $Q_{\text{rel}}(s_0, a', a)$ is directly known.

3.2. Handling inexact values

According to (5), we require the value $x_{i,j}$ as learning information for our relative Q function. Due to our limited feedback in the form of $x_{i,j} > 1$, we cannot determine the exact value. Therefore, we are purposely underestimating $x_{i,j}$ (as $\tilde{x}_{i,j}$), meaning that $Q(s_0, a) \geq Q(s_0, a') \cdot \tilde{x}_{i,j} \Leftrightarrow Q(s_0, a) = Q(s_0, a') \cdot (\tilde{x_{i,j}} + k_{i,j}), k_{i,j} \in \mathbb{R}^+$ for $r : S \times A \to \mathbb{R}^+$. This way, we can simply set $\tilde{x}_{i,j} = 1$ for describing "at least slightly better". This also means $x_{i,j} = \tilde{x}_{i,j} + k_{i,j}$ and $Q_{\text{rel}}(s_0, a, a') = \tilde{Q}_{\text{rel}}(s_0, a, a') + k_{i,j}$ in terms of the estimated value $\tilde{Q}$. Inserting this into (6) yields

$$\begin{aligned}
&Q_{\text{rel}}(s_0, a', a) = Q_{\text{rel}}(s_0, a, a') \cdot -\frac{1}{x_{i,j}} \\
\Leftrightarrow &\, Q_{\text{rel}}(s_0, a', a) = (\tilde{Q}_{\text{rel}}(s_0, a, a') + k_{i,j}) \cdot -\frac{1}{\tilde{x_{i,j}} + k_{i,j}} \\
\Leftrightarrow &\, Q_{\text{rel}}(s_0, a', a) = (\tilde{x_{i,j}} - 1 + k_{i,j}) \cdot -\frac{1}{\tilde{x_{i,j}} + k_{i,j}}
\end{aligned} \tag{7}$$

We have to consider that (6) is not applicable anymore, because the true values Q_{rel} and $x_{i,j}$ are unknown. We use the inverse estimate $Q_{\text{rel}}(s_0, a, a') = \tilde{Q}_{\text{rel}}(s_0, a, a') \cdot -\frac{1}{\tilde{x_{i,j}}} = -\frac{\tilde{x_{i,j}}-1}{\tilde{x_{i,j}}}$, which is an overestimate. We can now see that this estimate is in fact an overestimate by proving the relation to the correct estimate (7).

$$\begin{aligned} -\frac{\tilde{x_{i,j}}-1}{\tilde{x_{i,j}}} &\geq -\frac{\tilde{x_{i,j}}-1+k_{i,j}}{\tilde{x_{i,j}}+k_{i,j}} \\ \Leftrightarrow -(\tilde{x_{i,j}}-1)(\tilde{x_{i,j}}+k_{i,j}) &\geq -(\tilde{x_{i,j}}-1+k_{i,j})\cdot\tilde{x_{i,j}} \\ \Leftrightarrow -\tilde{x_{i,j}}^2-\tilde{x_{i,j}}k_{i,j}+\tilde{x_{i,j}}+k_{i,j} &\geq -\tilde{x_{i,j}}^2+\tilde{x_{i,j}}-\tilde{x_{i,j}}k_{i,j} \\ \Leftrightarrow k_{i,j} &\geq 0 \end{aligned} \tag{8}$$

Of course, (8) is always true because of $k_{i,j} \in \mathbb{R}^+$. For determining the best approximation $\tilde{Q}_{\text{rel}}(s,a,a')$, we can now average the directly determined underestimates and the overestimates derived from $\tilde{Q}_{\text{rel}}(s,a',a)$.

Using $r : S \times A \to \mathbb{R}^-$, this becomes an average over directly determined overestimates and derived underestimates, due to $x_{i,j} = \tilde{x}_{i,j} - k_{i,j}$.

3.3. Estimating subsections

According to the every-visit Monte Carlo method, we do not only want to update $Q(s_0,a)$, but also intermediate state-action pairs. Thus, we require an estimate for $y_{i,j}$ s.t. $V(T_i^{\vdash}) = V(T_j^{\vdash})\cdot y_{i,j}$. This is the relative value of the latter part of each trajectory from a predefined split point. $\bar{r}_T$ denotes the mean of rewards in the trajectory with r_t^Δ as the deviation of the reward $r(s_t,a_t) = \bar{r}_T + r_t^\Delta$. Considering (2) and the low variance assumption (Section 2.3), yields

$$\begin{aligned} V(T) = \sum_{t=0}^{|T|-1}\gamma^t r(s_t,a_t) \quad &= \sum_{t=0}^{|T|-1}\gamma^t(\bar{r}_T + r_t^\Delta) \\ \approx \sum_{t=0}^{|T|}\gamma^t\cdot\bar{r}_T & \end{aligned} \tag{9}$$

This is a geometric series and therefore equivalent to $V(T) \approx \bar{r}_T\frac{1-\gamma^{|T|+1}}{1-\gamma}$. Additionally, we define the relative difference $d = \frac{|T^{\vdash}|}{|T^{\dashv}.T^{\vdash}|}$.

$$\begin{aligned} V(T_i) &= V(T_j)\cdot x_{i,j} \\ \Rightarrow \bar{r}_{T_i}\frac{1-\gamma^{|T_i|+1}}{1-\gamma} &\approx \bar{r}_{T_j}\frac{1-\gamma^{|T_j|+1}}{1-\gamma}\cdot x_{i,j} \end{aligned} \tag{10}$$

$$\begin{aligned} V(T_i^{\vdash}) &= V(T_j^{\vdash})\cdot y_{i,j} \\ \Rightarrow \bar{r}_{T_i}\frac{1-\gamma^{d_i|T_i|+1}}{1-\gamma} &\approx \bar{r}_{T_j}\frac{1-\gamma^{d_j|T_j|+1}}{1-\gamma}\cdot y_{i,j} \end{aligned} \tag{11}$$

By expanding (11) and inserting (10) we can determine the value of $y_{i,j}$ relative to the known value $x_{i,j}$.

$$\begin{aligned}
&\bar{r}_{T_i}\frac{1-\gamma^{d_i|T_i|+1}}{1-\gamma} \approx \bar{r}_{T_j}\frac{1-\gamma^{d_j|T_j|+1}}{1-\gamma}\cdot y_{i,j} \\
\Leftrightarrow\ &\bar{r}_{T_i}\frac{1-\gamma^{d_i|T_i|+1}}{1-\gamma}\frac{1-\gamma^{|T_i|+1}}{1-\gamma} \\
&\quad\approx \bar{r}_{T_j}\frac{1-\gamma^{d_j|T_j|+1}}{1-\gamma}\frac{1-\gamma^{|T_i|+1}}{1-\gamma}\cdot y_{i,j} \\
\Leftrightarrow\ &\bar{r}_{T_j}\frac{1-\gamma^{d_i|T_i|+1}}{1-\gamma}\frac{1-\gamma^{|T_j|+1}}{1-\gamma}\cdot x_{i,j} \\
&\quad\approx \bar{r}_{T_j}\frac{1-\gamma^{d_j|T_j|+1}}{1-\gamma}\frac{1-\gamma^{|T_i|+1}}{1-\gamma}\cdot y_{i,j} \\
\Leftrightarrow\ &\frac{(1-\gamma^{d_i|T_i|+1})(1-\gamma^{|T_j|+1})}{(1-\gamma^{d_j|T_j|+1})(1-\gamma^{|T_i|+1})}\cdot x_{i,j} \approx y_{i,j}
\end{aligned} \tag{12}$$

(12) defines the relative difference for intermediate states, which can be used to update the relative Q-Function, as described in section 3.1. It should be noted, this estimate is only calculated for states occurring in both trajectories. Different states cannot be compared without some kind of distance or similarity function, introducing new requirements for the state space definition.

Of course, according to section 3.2, we must assume $\tilde{x}_{i,j}+k_{i,j}$ to be the true reward factor, hence $y_{i,j} \approx z_{i,j}\cdot(\tilde{x}_{i,j}+k_{i,j})$ with $z = \frac{(1-\gamma^{d_i|T_i|+1})(1-\gamma^{|T_j|+1})}{(1-\gamma^{d_j|T_j|+1})(1-\gamma^{|T_i|+1})}$. But again, due to $k_{i,j}\in\mathbb{R}^+$, we know $y_{i,j} \approx z_{i,j}\cdot(\tilde{x}_{i,j}+k_{i,j}) \geq z_{i,j}\cdot\tilde{x}_{i,j}$ which means $z_{i,j}\cdot\tilde{x}_{i,j}$ is an underestimation of the true value. We are omitting $y_{i,j}<1$ for $r: S\times A\to\mathbb{R}^+$ ($y_{i,j}>1$ for $r: S\times A\to\mathbb{R}^-$), as this would be in fact a training information contradicting the current preference, but it is probably an estimation error.

4. The Every Visit Preference Monte Carlo Algorithm (EPMC)

We have now the means to define a every-visit Monte Carlo policy iteration algorithm for preference feedback. A fixed amount of trajectories is sampled in each iteration, according to an ϵ-greedy strategy (Sutton and Barto, 1998) After determining the preferences for all trajectory pairs of the current iteration, the relative Q-values $Q_{\text{rel}}(s_0,a,a')$ are computed (section 3.1 & 3.2). Of course, these values are in fact only Monte Carlo estimates and averaged to determine the true approximation, that is used to update the Q function. This is also performed for intermediate states, after calculating the estimate as described in sec. 3.3. The Q-function is updated with a common learning rate dependent increase in the form of $Q_{\text{rel }i+1}(s,a,a') = (1-\alpha)Q_{\text{rel }i}(s,a,a') + \alpha\tilde{Q}^{\pi_i}_{\text{rel}}(s,a,a')$ with $\tilde{Q}^{\pi_i}_{\text{rel}}(s,a,a')$ as the approximation gained after i iterations. The resulting optimal policy is defined by (13).

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_{a\in A(s)} \frac{1}{|A(s)|-1} \\ & \qquad\cdot \sum\limits_{a'\in A(s), a\neq a'} Q_{\text{rel}}(s,a,a') \\ 0 & \text{else} \end{cases} \tag{13}$$

Of course, the maximum can only be assumed to be the best action if $r : S \times A \rightarrow \mathbb{R}^+$. In this case of $r : S \times A \rightarrow \mathbb{R}^-$ where small values are preferable, we store $Q_{\text{rel}}(s_0, a, a') = \frac{1}{\tilde{x}_{i,j}} - 1$ instead of $\tilde{x}_{i,j} - 1$, because this allows us to reuse the maximum as optimal policy and does not affect the outcome due to the absolute value not being relevant.

A pseudo-code of the resulting algorithm is shown as algorithm 2. The initial policy π_0 is the random policy. In each iteration, n trajectories are sampled with the ϵ-greedy strategy. These trajectories are compared in a pairwise manner to determine $\frac{n(n-1)}{2}$ preferences. All possible Q_{rel}-value estimates are then calculated and averaged for updating the policy, subject to the learning rate α.

```
input  : initial policy $\pi_0$, start states $S_0$, iteration limit $m$
         sample limit $n$, random probability $\epsilon$, learning rate $\alpha$
output: improved policy $\pi_m$

q-updates = $\emptyset$
for $i = 0$ to $m$ do
    $s_0$ = RANDOM($S_0$), trajs = $\emptyset$, prefs = $\emptyset$, q-averages = $\emptyset$
    for 0 to $n$ do
        trajs ← trajs ∪ $\epsilon$-GreedyRollout($s_0, \pi, \epsilon$)
    end
    prefs ← determinePreferences(trajs)
    q-updates ← q-updates ∪ createEstimates(trajs,prefs)
    q-averages ← createAverages(q-updates)
    $\pi_{i+1}$ = updatePolicy($\pi_i$, q-averages, $\alpha$)
end
```

Algorithm 2: EPMC

5. Experimental Setup

For evaluating our algorithm, we picked three common benchmark problems: mountain car and acrobot with $r : S \times A \rightarrow \mathbb{R}^-$ as well as inverted pendulum for $r : S \times A \rightarrow \mathbb{R}^+$. The parametrization from Sutton and Barto (1998) was used for the first two problems and Wirth and Fürnkranz (2013) for the inverted pendulum, but with a 20% transition noise for all three domains. The mountain car setup is additionally subject to random start states. We use a tabular state representation based on a discretization of each state variable into 10 bins. In each trial, we performed $m = 50$ policy iterations using $n = 10$ trajectories in each iteration. For the inverted pendulum m was lowered to 20, due to the faster convergence rates on this problem. The horizon is always 500. γ was set to 0.99 with ϵ and α set to the best values found by a 10×10 uniform grid search. In general, all results presented are averages over 100 consecutive trials. Besides of the application of EPMC with the mentioned binary preference (*EPMC Binary*), we also evaluated the loss compared to the true rewards by not using the underestimate $\tilde{x}_{i,j}$ as described in sec. 3.2, but the true value $x_{i,j}$ (*EPMC True*). As the value of $x_{i,j}$ is in practice unknown, these result may serve as an upper bound on the achievable performance.

Because of the close relation to the algorithm defined by Wirth and Fürnkranz (2013), we also preformed the mentioned experiments with an implementation of this algorithm as baseline comparison (*Baseline*). The parameters η and α are also determined by a 10×10 uniform grid search. For convenience, the approach is described in the following section 5.1.

We also compared the algorithm to SARSA(λ) with an ϵ-greedy sampling strategy, because it is arguably the most common algorithm for solving reinforcement learning problems. Its hyperparameters λ, α and ϵ are determined by a $10 \times 10 \times 10$ grid search. The amount of trajectory samples is identical to the preference based setups, which already results in much more valuable feedback. A numeric value enables a comparison to *all* other trajectories, not only to the n samples of the current iteration, as performed by EPMC. This means EPMC is utilizing $\frac{n(n-1)}{2} \cdot m = 2250$ preferences (900 for inverted pendulum, respectively) while the real-valued feedback obtained by SARSA(λ) could be used to compute $\frac{nm(nm-1)}{2} = 112250$ preferences (19900 respectively). But due to the completely different method, we decided to use the amount of sampled trajectories $n \cdot m$ as fixed dimension for the comparison.

5.1. The Wirth and Fürnkranz (2013) algorithm

This algorithm also uses every-visit Monte Carlo estimates, but the relative difference is not stored in terms of a Q-function, but as probability of an action being preferred over another one. This probability is calculated based on the number n of states occurring in both compared trajectories T_1 and T_2

$$\Pr(a \succ a'|s, T_i \succ T_j) = \frac{n+1}{2n} \tag{14}$$

where $(s, a) \in T_i$ and $(s, a') \in T_j$.

The idea behind this estimate is, if it is possible to assume that only action choices in these overlapping states are relevant for the (relative) outcome, then the outcome is related to the ratio of preferred actions chosen. These samples are aggregated and used to determine the probability of an action being the most preferable one, as described by (15).

$$\Pr(a|s) = \frac{1}{\sum\limits_{a' \in A(s), a' \neq a} \frac{1}{\Pr(a \succ a'|s)} - (k-2)}. \tag{15}$$

This value is deemed comparable to the reward of a classic MDP. By modifying the EXP3 exploration/exploitation strategy (Audibert and Bubeck, 2009) accordingly results in (16) as action selection probability for sampling new trajectories. $\eta \in (0, 1]$ is a user settable parameter.

$$\tilde{g}_{s,a,i} = \frac{\Pr^{\pi_i}(a|s)}{\text{EXP3}(s,a)_i} \mathbf{1}_{s \in P_s^{\pi_i}} \quad \tilde{G}_{s,a,i} = \sum_{t=1}^{i} \tilde{g}_{s,a,t}$$

$$\text{EXP3}(s,a)_{i+1} = \frac{\eta}{|A|} + (1 - |A(s)|\frac{\eta}{|A|}) \frac{\exp(\frac{\eta}{|A|}\tilde{G}_{s,a,i})}{\sum\limits_{b \in A(s)} \exp(\frac{\eta}{|A|}\tilde{G}_{s,b,i})} \tag{16}$$

This strategy is only applied to the first trajectory sampled in each policy iteration step and for all states overlapping with a trajectory already seen in this evaluation step. All other actions are selected according to a approximate best policy

$$\tilde{\pi}(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_{a'}(\Pr(s|a')) \\ 0 & \text{else.} \end{cases} \tag{17}$$

This enables the assumption, that only overlapping states are relevant for the (relative) outcome due to all other action choices already being optimal.

6. Results

All configurations are analyzed with respect to two different metrics, the quality of the policy after the last iteration (*Policy*) and the learning curve (LC). The quality value is the amount of steps required to reach the goal. In the case of mountain car, this is an average over 100 randomly sampled start states. The learning curve metric is the sum of the quality of the policy after each iteration, meaning a higher value correlates with faster convergence. The values for mountain car and acrobot are negative, because here each step is penalized, as opposed to the step reward case for the inverted pendulum experiments.

6.1. Performance

In the inverted pendulum domain (Table 1), EPMC outperforms the baseline algorithm by a sizeable margin for both metrics. It results in policies comparable to the SARSA(λ) results, but has a somewhat slower convergence. Table 2 shows the results for the acrobot domain.

Configuration	Policy	LC
Baseline	345.64	45287.8
Sarsa	474.63	91012.7
EPMC Binary	469.845	74314.2
EPMC True	478.865	77420.9

Table 1: Results for inverted pendulum

Configuration	Policy	LC
Baseline	-300.32	-186714.8
Sarsa	-210.25	-134223.2
EPMC Binary	-192.16	-138587.0
EPMC True	-189.41	-135526.7

Table 2: Results for acrobot

Here, the baseline algorithm and SARSA(λ) are not able to reach a policy as good as the one found by EPMC. The improvement over the baseline is again very substantial, but this time the convergence rate is close to the one achieved by SARSA(λ). In the mountain car

Configuration	Policy	LC
Baseline	-68.64	-57598.5
Sarsa	-56.24	-39796.4
EPMC Binary	-60.94	-50677.1
EPMC True	-59.97	-51997.1

Table 3: Results for mountain car

domain (Table 3), EPMC also outperforms the baseline algorithm by a large margin but it could not reach the performance of SARSA(λ) for these experiments. Still the difference is comparably small, especially in terms of the final policy. In summary, we observe that the performance of the preference-based algorithms is quite close to the performance of SARSA(λ), but with a somewhat slower convergence rate. This is not surprising because SARSA(λ) is able to use the numerical values of the rewards, while the other algorithms only make use of the order relation between the rewards, which is induced by the observed preferences. As expected, *EPMC True*, which makes use of an oracle for the correct relative adjustment, outperforms the binary preference version, but the differences are quite small, which indicates that the loss due to the estimation error (Section 3.2) is small.

6.2. Convergence

In the following, we show the convergence graphs of the best configuration found for all experiments, excluding the baseline algorithm. In the pendulum domain (Figure 1), we can clearly see the the stated convergence advantage of SARSA. The slight improvements of the EPMC true over EPMC binary are also visible. Figure 2 shows the results for the

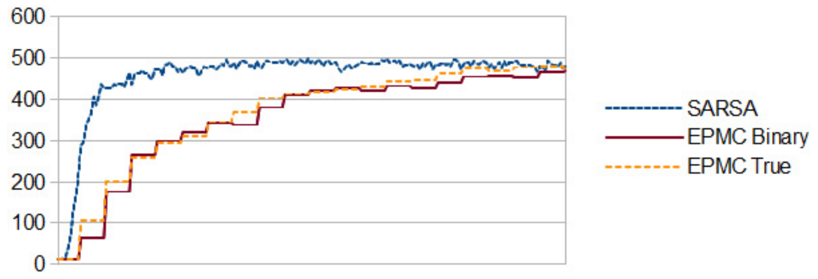

Figure 1: Convergence in the inverted pendulum domain

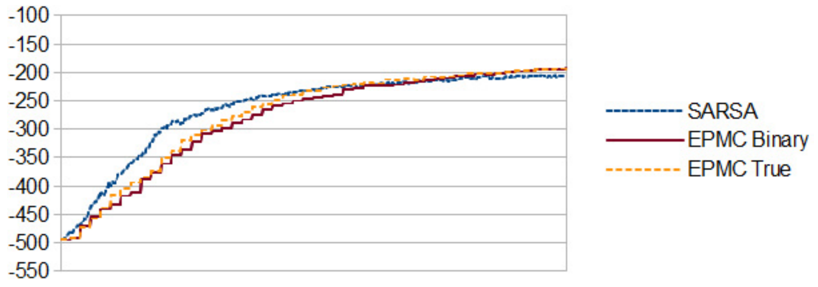

Figure 2: Convergence in the acrobot domain

acrobot domain. The graph also confirms the performance results. All three algorithms show similar behavior with only a minor advantage for SARSA. Finally, the mountain car graph (Figure 3) is similar to Figure 1 (inverted pendulum). All algorithms show continuous convergence to the optimum, but SARSA is again converging faster, especially for the first trajectories.

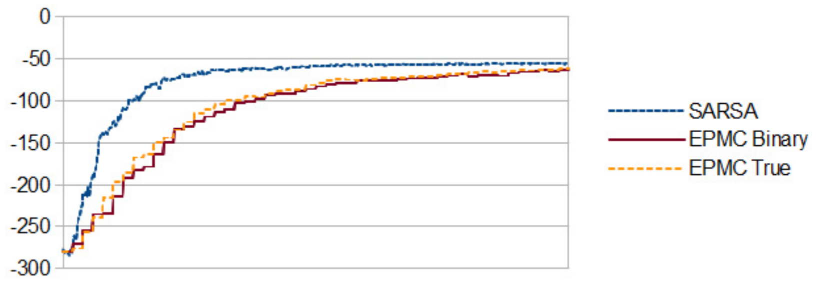

Figure 3: Convergence in the mountain car domain

7. Future Work

Of course, the most important next step is relaxing the variance condition, because this is the most restrictive limitation that has to be considered. We also hope to eliminate the required distinction between a positive and a negative reward space, which would not only make the application of the algorithm easier but would also enable a broader range of domains where the algorithm can be applied. We also expect to be able to increase the performance by the use of better sampling strategies, a topic closely related to active learning (Settles, 2009).

We also aim at applying this algorithm to problems that can not be described by a single objective, requiring the addition of concepts like Pareto optimality. This would enable the application to problems that are not solvable with quantitative methods.

8. Related Work

As mentioned, this paper is comparable to the work of Wirth and Fürnkranz (2013). This algorithm uses a probability theorem, instead of the suggested preference estimate.

The work of Fürnkranz et al. (2012) is also related. They also identify action preferences, but do not reuse trajectories for creating learning information for intermediate state-action pairs, meaning the preference feedback is used less efficient. Additionally, the problem is cast as a pairwise label ranking problem with the state features as attributes and the action as class labels. This enables a generalization to unseen states by training a learning algorithm like multilayer perceptrons for predicting a ranking of the actions in each state.

Akrour et al. (2012) use an *evolutionary strategy (ES)* for creating new policies within a parameterized policy space. Each trajectory is evaluated by the amount of occurring *sensori-motor-state (SMS)* cluster. An SMS is a vector of all sensor and actuator values of the underlying robotics system. The utility of a trajectory is defined as a weighted, linear sum over these clusters with the weights learned by an SVM$^{\text{Rank}}$ like approach (Joachims, 2002) that can utilize pairwise preference information. The utility of a policy can then be estimated by the weight vector and sampled occurrences of SMS clusters. This enables the ranking of policies for creating new policies within the ES framework.

The *Bayesian preference learning from trajectory preference queries (BayesTPQ)* algorithm (Wilson et al., 2012) also requires a parameterized policy space. By utilizing a trajectory distance function, it is possible to estimate a probably distribution over policies, based on pairwise preferences. This probability is maximized for creating the best policy concerning the given preferences.

9. Conclusions

EPMC shows that Monte Carlo based approaches to preference-based reinforcement learning are very promising and already able to compete with SARSA(λ). This is especially interesting, because preferences are lacking a great amount of information included in traditional rewards. Not only the numeric grading is missing, but the feedback is also only available for certain trajectory pairs, wheras the conventional numeric feedback allows a comparison between all trajectories. Current PBRL algorithms still offer plenty of room for improvements, as shown by the comparison to the baseline algorithm of Wirth and Fürnkranz (2013). We have also shown, that it is possible to use estimates for the numeric preference grade without loosing substantial information. The greatest drawback of the EPMC algorithm is its variance condition. Many domains do conform to this requirement, but relaxing it would greatly increase the amount of possible applications.

10. Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Priority Programme 1527. We would like to thank the reviewers for helpful suggestions.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21th international conference on Machine learning (ICML-04)*, page 1. ACM, 2004.

Abbeel, P. and Ng, A. Y. Inverse reinforcement learning. In *Encyclopedia of Machine Learning*, pp. 554–558. Springer, 2010.

Akrour, R., Schoenauer, M., and Sebag, M. Preference-based policy learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11)*, volume 6911 of *LNCS*, pages 12–27. Springer, 2011.

Akrour, R., Schoenauer, M., and Sebag, M. APRIL: Active preference learning-based reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-12)*, volume 7524 of *LNCS*, pages 116–131. Springer, 2012.

Audibert, J.-Y. and Bubeck, S. Minimax policies for adversarial and stochastic bandits. In *Proceedings of the 22nd Conference on Learning Theory (COLT-09)*, pages 773–818. Omnipress, 2009.

Fürnkranz, J., Hüllermeier, E., Cheng, W. and Park, S.-H. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89 (1-2), pages 123–156. Springer, 2012.

Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 133–142. ACM Press, 2002.

Settles, B. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Sutton, R. S. and Barto, A. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

Wilson, A., Fern, A. and Tadepalli, P. A bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems 25: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-12)*, pages 1142–1150. Curran Associates, 2012.

Wirth, C. and Fürnkranz, J. A policy iteration algorithm for learning from preference-based feedback. In *Advances in Intelligent Data Analysis XII: 12th International Symposium (IDA-13)*, volume 8207 of *LNCS*. Springer, 2013.

Wirth, C. and Fürnkranz, J. Preference-based reinforcement learning a preliminary survey. In *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards*, 2013.

Zhao, Y., Kosorok, M. and Zeng, D. Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28(26), pages 3295–3315, 2009.